A Framework for Software Architecture Visualisation Assessment

K. Gallagher A. Hatch & M. Munro
Visualisaton Research Group
Computer Science Department, University of Durham, South Road
Durham DH1 3LE, UK
{k.b.gallagher|a.s.hatch|malcolm.munro}@durham.ac.uk

Abstract

In order to assess software architecture *visualisation strategies, we qualitatively characterize then construct an assessment framework with 7 key areas and 31 features. The framework is used for evaluation and comparison of various strategies from multiple stakeholder perspectives. Six existing software architecture visualisation tools and a seventh research tool were evaluated. All tools exhibited shortcomings when evaluated in the framework.*

1 Introduction and Related Work

1.1 Architecture

In the IEEE 1471-2000 standard [8], architecture is defined as "the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution." Architecture can take two roles: one describing how the software system's architecture should be; the other role describing how a software system's architecture is.

While there are many variations on the theme of software architecture, for the purposes of this work work we use "Software architecture is a representation of a software system at its highest level of abstraction, consisting of a set of fundamental building blocks for the software system and their interconnection." [6].

1.2 Visualisation

Software visualisation attempts to retrieve and present information about a software system to a user in a visual format. By doing so, the user is often able to understand the information presented in a shorter period of time, or to a greater depth.

Visualisation, the process, can refer to the activity that people undertake when building an internal picture about real world or abstract entities. The process also includes decisions on metaphors, environment and interactivity. The term visualisation in this work describes the process of mapping entities in the system domain to graphical representations.

1.3 Stakeholders

For any software system, there are a number of *stakeholders*, each with differing perspectives and requirements of a software architecture depending on their role. Stakeholders include the architects; designers; developers; the sales, services and support teams; and the customer.

Communication and understanding of the *architecture* is essential in ensuring that each stakeholder can play their role during a system's lifetime. In supporting mainly developers and maintainers, software visualisation has been largely concerned with representing static and dynamic aspects of software at the code level. *Architecture* visualisatons require a larger set of stakeholders.

1.4 Software Visualisation

Price, et al., [11] presented a taxonomy of software visualization. Our technique is similar to theirs in that we also use a phenomenological approach to derive properties from existing tools, then generalize to a framework. Bassil and Keller [3] use Price's framework to qualitatively analyze a collection of *software* visualisation tools. Maletic, et al., [10] enhance the Price framework to regard *task orientation*.Hatch [6] surveys software visualisation as a prelude to software *architecture* visualisation.

1.5 Software *Architecture* Visualisation Requirements

The aim of this work is to explore the ways in which software *architecture* visualisation can assist in the tasks undertaken by the differing stakeholders in a software system. A comprehensive approach to software architecture visualisation that is based on soft-

ware visualisation techniques should meet the following requirements:

- support multiple representations of software architecture.
- support multiple stakeholders of software architecture.
- support both static data and dynamic data.
- utilise a flexible data model to allow for the capture of a wide spectrum of architectures.
- utilise a flexible render model that allows for the creation of many graphical components.
- utilise a flexible renderer for defining different views.

The software architecture visualisation requirements can be incorporated into an evaluation framework.

1.6 Result Summary

Our work is similar to that of Storey, et al. "[T]he role of our framework is to act primarily as a *discussion tool*. [The framework] can serve several purposes: 1) as a *formative evaluation* tool... 2) for potential *tool users*...; and 3) as a *comparison tool*..." [16] The principal difference being that this work is about *architecture*, while theirs is about *development*.

2 Visualisation Evaluation Strategies

There are several strategies adopted when evaluating software visualisations design guidelines and feature-based evaluation frameworks. Design guidelines are informal statements about aspects of a field that are intended to assist further inquiry. Informal statements promote reasoning about aspects of the visualisation to quickly determine their value. Guidelines can identify the relative merit of cognitive issues. The primary use of design guidelines is to help in early stage evaluation.Identifying areas in which guidelines are used as evaluations can help in identifying where there is a lack of evaluation frameworks.

Feature-based frameworks often take on the form of multiple-choice questions which can be answered comparatively quickly. The application of an evaluation framework imposes no prerequisites on infrastructure or target system [9] and allow for evaluation of a system at multiple levels of detail. Frameworks therefore facilitate an evaluation capability with low overhead and investment. Here, the style of question used is of critical importance. Framework questions that result in simple "yes" or "no" answers may be too open ended. Other problems with questions include a sliding scale, which may then become too subjective, or questions that depend very much on user experience. Finally, some questions are simply too vague to be answered accurately.

3 Evaluation Framework

3.1 Rationale for the Evaluation Framework

A new framework is required when considering the evaluation of visualisations that explicitly address software at an *architectural* level.Software visualisations do capture several aspects of visualisation that are directly applicable to software architecture visualisation. The framework described here is not without a strong basis in software visualisation evaluation.

We use a design guideline approach to build a feature-based framework. The modular structure described allows individual concerns to be addressed in comparative isolation, and so the application of the framework need not be performed in its entirety. Some software visualisations make no attempt to evaluate the implementation of a visualisation, but focus on the visualisation itself. This framework also considers implementation, because some properties of a visualisation can be better reasoned about in terms of its implementation.

3.2 Description of the Evaluation Framework

The proposed framework is divided into seven key areas: static representation, dynamic representation, views, navigation & interaction, task support, implementation and visualisation. Static representation characterizes the size and accessibility of the architectural information. Dynamic representation characterizes the support for run-time collection and observation of architectural information. Views characterize the perspective of the observer. Navigation and Interaction characterize the ease of use of the tool. Task support characterizes the operational use of the visualisation. Implementation assesses the "computableness" the information. And visualisation characterizes the quality of the information presented to the observer. Visualisation refers to hardware; views refer to the observer. While it may be that some of the following items could be shifted from one key area to another, the point is that any framework must address all of these concerns. Table 1 presents a summary of the individual evaluation framework questions/items. Table 2 is the set of possible responses to questions in the key areas.

Key Area	Question
Static Representation (SR)	
SR 1	Multiple software architectures
SR 2	Types of software architecture
SR 3	Recovery of software architecture information
SR 4	Accommodate large volumes of information
Dynamic Representation (DR)	
DR 1	Support dynamic data
DR 2	Associate events with architectural elements
DR 3	Non invasive approaches
DR 4	Live collection
DR 5	Replay data
Views (V)	
V 1	Multiple views
V 2	Representation of viewpoint definition
Navigation and Interaction (NI)	
NI 1	Browsing
NI 2	Searching
NI 3	Query building
NI 4	Inter-view navigation
NI 5	View navigation
Task Support (TS)	
TS 1	Represent anomalies
TS 2	Comprehension
TS 3	Annotation
TS 4	Communication
TS 5	Show evolution
TS 6	Construction
TS 7	Planning and execution
TS 8	Evaluation
TS 9	Comparison
TS 10	Show rationale
Implementation (I)	
I 1	Automatic generation
I 2	Platform dependence
I 3	Multiple users
Visualisation (VN)	
VN 1	High fidelity and completeness
VN 2	Dynamically changing architecture

Table 1: Evaluation Framework

Response	Meaning
Y	Full support
Y?	Mainly supported
N?	Mainly not supported
N	No support
NA	Not Applicable
?	Unable to determine

Table 2: Possible responses to items in Table 1

3.2.1 Static Representation (SR)

A visualisation should be capable of accessing its data source. Oftentimes, architectural data often does not reside in a single location and must be extracted from a multitude of sources. An architecture visualisation certainly benefits from the ability to support the recovery of data from a number of disparate sources. Moreover, with multiple data sources there should be a mechanism for ensuring that the data can be consolidated into a meaningful model for the visualisation.

Architectural information may be recovered from sources that are non-architectural. File-systems packages, classes, methods and variables can all contribute to a view of the software architecture, and so a visualisation system should support these data types.

If architectural data is to be retrieved from non-architectural data, there is a potential for the data repository to contain large amounts of data from lower levels of abstraction. If this is the strategy employed by the visualisation, then the visualisation should be able to deal with large volumes of information; i.e., the system should be scalable.

3.2.2 Dynamic Representation (DR)

Runtime information can indicate a number of aspects of the software architecture. Visualisations should support the collection of runtime information from dynamic data sources in order to relay runtime information. When dynamic events occur, the visualisation should be able to display these events appropriately, and within the context of the architecture. The visualisation must therefore be able to associate incoming events with architectural entities.

Disruptive behaviour is not desirable. The visualisation system should support a suitable approach to recovery of dynamic architecture data in the least-invasive way.

3.2.3 Views (V)

A visualisation may support the creation of a number of views of the software architecture, and may wish to allow simultaneous access to these views. In the IEEE

1471 standard [8], architectural views have viewpoints associated with them. A viewpoint defines a number of important aspects about that view including the stakeholders and concerns that are addressed by that viewpoint, along with the language, modeling techniques and analytical methods used in constructing the view based on that viewpoint. A visualisation may choose to make this information available to the user in order to assist in their understanding of the view they are using.

3.2.4 Navigation and Interaction (NI)

Storey, et al. [15] indicate that a software visualisation system should provide directional navigation. An architectural visualisation should support the same facility.

Searching is the data-space navigation process that allows the user to locate information with respect to a set of criteria. Storey, et al [15] label this as arbitrary navigation – being able to move to a location that is not necessarily reachable by direct links.

Query building is a hybrid combination of browsing and searching. It allows a user to find a set of information, and then continually expand on a search in a particular direction by repeated searching from a related result. Visualisations should support this style of data-space navigation.

Context should also be maintained when switching between views so as to reduce disorientation. Along with data-space navigation, the movement within a view is also important. Schneiderman's mantra for visualisation is overview first, zoom and filter, and then show details on demand [12]. A visualisation system should support this strategy. Also, the visualisation should allow the user to move around so as to focus on and see the information they are looking for.

3.2.5 Task Support (TS)

The visualisation should be able to cope with data anomalies that are unexpected and may cause unwanted behaviour and report these anomalies to the user if they can be detected.

Stakeholders may require very different views from other stakeholders. As comprehension strategies are task dependent, architecture visualisations should support either of top-down or bottom-up strategies, or a combination of the two.

An architecture visualisation should provide a facility to show the evolution. This support may be basic, showing architectural snapshots, or the support may be more advanced by using animation.

Visualisations may offer the capability for the users to create, edit and delete objects in the visualisation. Architectural descriptions can be used for the planning, managing and execution of software development [8]. In order for the visualisation to support this task, it should provide rudimentary functionality of a project management tool – or have the ability to communicate with an existing project management tool.

Software architecture evaluation allows the architects and designers to determine the quality of the software architecture and to predict the quality of the software that conforms to the architecture description [8]. A visualisation should have some mechanism by which quality descriptions can be associated with components of the software being visualised.

Rationale for the selection of architecture, and the selection of the individual architectures of the components of that architecture, are included in architectural descriptions. Rationale can also be associated with each viewpoint of an architecture.

3.2.6 Implementation (I)

Visualisations need to be able to be generated automatically. A visualisation should be able to execute on a platform suitable for the types of software it is intended to visualise. As there are many stakeholder roles in a software system, there may also be a one-to-one mapping of role to physical users. Therefore the visualisation should support multiple users concurrently, or asynchronously.

3.2.7 Visualisation (VN)

For software architecture visualisation, the visualisation must present the architecture accurately and completely. During its execution, software may change its configuration in such a way that its architecture has changed. Dynamic architecture characterises software that changes its architecture as it executes. If the visualisation is able to support architectural views of the software at runtime, then it may be capable of showing the dynamic aspects of the architecture.

4 Tools

This section presents a brief summary and discussion of the features of tools that are to be assessed.

4.1 ArchView (AV)

The ArchView [4] tool uses the architecture analysis activities of extraction, visualisation and calculation. It produces an architecture visualisation that presents the *use* relations in software systems.

4.2 The Searchable Bookshelf (SB)

The Searchable Bookshelf [14] visualisation attempts to combine both searching and browsing approaches to software comprehension. The Searchable Bookshelf adds search capabilities to the Software Bookshelf. User to browse the software structure from an initial overview by navigating through an HTML style display and a software landscape central view.

4.3 SoftArch (SA)

SoftArch [5] is both a modeling and visualisation system for software, allowing information from software systems to be visualised in architectural views. SoftArch supports both static and dynamic visualisation of software architecture components, and does so at various levels of abstraction.

4.4 SoFi

SoFi [7] is a tool that performs source code analysis in order to compare intended architecture with implemented architecture. SoFi's clusters source files into a structure based on source file naming schemes. SoFi relies heavily on the intervention by an architect.

4.5 LePUS

LePUS is a formal language dedicated to the specification of object-oriented design and architecture [1, 2]. LePUS diagrams are intended to be used in the specification of architectures and design patterns, and in the documentation of frameworks and programs.

4.6 Enterprise Architect (EA)

Enterprise Architect [13] is a UML CASE tool that allows software architects, designers and analysts to design software from several viewpoints. EA can be used from requirements capture to UML modeling to testing and project management.

4.7 ArchVis (AVis)

ArchVis [6] is prototype software architecture visualisation tool. Its design was driven by the key concerns identified in section 1.5. That is to say that ArchVis was designed and built using the evaluation framework as requirements. In this sense, including it in this list is skews the results. However, the framework and ArchVis were developed in parallel, so features were added to the framework after the design of ArchVis was complete.

5 Evaluations

Table 3 presents the evaluation of the features in tabular form. Most tools do reasonably well in static representation. Dynamic representation is another matter, as none of the surveyed tools have support for this key area. Most tools support multiple views; none support viewpoint definition. Navigation and interaction is supported by browsing in all tools. The Enterprise Architect is the only tool that has all of the searching, querying, and view navigation features. It seems that all tools are deficient in task support. This is mildly surprising as one would expect architecture tools to be closely allied with project management and IDE systems. It is also surprising to note that not all tools have automatic generation and multiple user support. And all tools support high fidelity visualisation, but none dynamically changing architectures.

With respect to ArchVis, it is worth noting that it does not meet the full set of requirements. It does not show evolution and give comparisons, and has only lightweight support for anomalies and construction. It does meet the dynamic representation criteria, and thus has one singular advantage over all the other tools.

6 Conclusion

Software architecture is gross structure of a system; as such it presents a different set of problems for visualisation than those of the visualising the software itself. We used existing tools to develop and present a framework for the assessment of the capabilities of software *architecture* visualisation tools. The intent of the framework is formulation, assessment, comparison and discussion. Once the framework was constructed, we used it to evaluate the 7 tools that were used in its formulation.

References

[1] A.Eden. Visualization of object-oriented architectures. In *Twenty-third International Conference on Software Engineering*, Toronto, Canada, 2001.

[2] A.Eden. LePUS: A visual formalism for object-oriented architectures. In *Sixth World conference on Integrated Design and Process Technology*, Pasadena, California, June 2002.

[3] S. Bassil and R. Keller. A qualitative and quantitative evaluation of software visualization tools. In *Proceedings of the Workshop on Software Visualization*, pages 33–37, 2001.

[4] L. Feijs and R. de Yong. 3D visualization of software architectures. *Communications of the ACM*, 41(12), Dec 1998.

[5] J. Grundy and J. Hosking. High-level static and dynamic visualisation of software architectures.

	AV	SB	SA	SoFi	LePUS	EA	AVis
	Static Representation (DR)						
1	Y	Y	Y	Y	Y	Y	Y
2	Y	Y	Y	Y	Y	Y	Y
3	Y?	Y?	N	Y?	NA	N	Y
4	Y	?	N?	Y	NA	Y	Y
	Dynamic Representation (DR)						
1	N	N	N?	N	NA	N	Y?
2	N	N	N?	N	NA	N	Y
3	N	N	N?	N	NA	N	Y
4	N	N	N?	N	NA	N	Y
5	N	N	N?	N	NA	N	Y
	Views (V)						
1	N	Y	Y	N	Y	Y	Y
2	N	Y?	N?	N	N	Y	Y?
	Navigation and Interaction (NI)						
1	Y	Y	Y	N?	NA	Y	Y
2	N	Y	N	N	NA	Y	Y
3	N	N	N	N	NA	Y	Y
4	N	Y	Y	N	NA	Y	Y?
5	Y	N?	Y	N	NA	Y	Y
	Task Support (TS)						
1	Y	?	N	Y	NA	N	N?
2	Y	Y	Y	Y	Y	Y	Y
3	N	N	Y	N	Y?	Y	Y?
4	Y	Y	Y	Y	Y	Y	Y
5	N	N?	N	N	Y?	N?	N
6	N	N	Y	N	Y	Y	N?
7	N	N	N	N	N	N	Y?
8	Y	Y?	N	Y	Y	Y	Y?
9	Y?	Y?	N	Y?	Y	N?	N
10	N	N?	N	N	N?	Y	Y?
	Implementation (I)						
1	Y	Y	N	Y	NA	N	Y
2	Y?	N?	Y	?	NA	N	Y
3	N	Y	Y	N?	NA	Y	Y?
	Visualisation (VN)						
1	Y	Y	Y	Y	NA	Y	Y?
2	N	N	N	N	NA	N	N?

Table 3: Evaluation Summary. See Table 2 for key.

In *IEEE Symposium on Visual Languages*, Seattle, Washington, Sept 2000.

[6] A. Hatch. *Software Architecture Visualisation.* PhD thesis, University of Durham, 2004.

[7] I.Carmichael, V. Tzerpos, and R. Holt. Design maintenance: Unexpected architectural interactions. In *International Conference on Software Maintenance*, 1995.

[8] IEEE. IEEE Recommended practice for architectural description of software intensive systems. Technical report, IEEE, Piscataway, N.J., 2000.

[9] B. Kitchenham and L. Jones. Evaluating software engineering methods and tools. part 1: The evaluation context and methods. In *Software Engineering Notes*, Jan 1996.

[10] J. Maletic, A. Marcus, and M. Collard. A task oriented view of software visualization. In *Proceedings of the IEEE Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2002)*, pages 32–40, 2002.

[11] Blaine A. Price, Ronald Baecker, and Ian S. Small. A principled taxonomy of software visualization. *J. Vis. Lang. Comput.*, 4(3):211–266, 1993.

[12] B. Schneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction.* Addison-Wesley, 1998.

[13] Sparx Systems. *Enterprise Architect.* http://www.sparxsystems.com.au.

[14] S.Sim, C. Clarke, R. Holt, and A. Cox. Browsing and searching software architectures. In *International Conference on Software Maintenance*, Oxford, England, Sept 1999.

[15] M. Storey, F. Fracchia, and H. Muller. Cognitive design elements to support the construction of a mental model during software exploration. *Journal of Software Systems*, 44, 1999.

[16] M. A Storey, D. Cubranic, and D. German. On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and Framework. In *SoftVis–05, ACM Symposium on Software Visualization*, 2005.